Towards an Approach for Orchestrating Design Space Exploration Problems to Fix Multi-Paradigm Inconsistencies

Sebastian J. I. Herzig, Benjamin Kruse, Federico Ciccozzi, Joachim Denil, Rick Salay, and Dániel Varró

sebastian.herzig@gatech.edu, bkruse@ethz.ch, federico.ciccozzi@mdh.se, joachim.denil@uantwerpen.be, rsalay@cs.toronto.edu, varro@mit.bme.hu

Abstract. In model-driven engineering, the aim of design space exploration (DSE) is to generate a set of design candidates that satisfy a given set of constraints and requirements, and are optimal with respect to some criteria. In multi-paradigm modeling it is not uncommon to perform a variety of computationally expensive analyses as part of such an exploration process. However, existing DSE techniques do not take dependencies between solver operations into account, thereby failing to provide a mechanism for pruning infeasible solutions early. Motivated by this source of inefficiency, this paper discusses a conceptual approach to orchestrating solvers and design space exploration problems.

Keywords: design space exploration, solver orchestration

1 Introduction

The design of complex systems necessitates the exploration and evaluation of design alternatives from different perspectives leveraging multi-paradigm modeling languages and tools. Different functionally equivalent design candidates need to be systematically sought out and derived by Design Space Exploration (DSE) techniques [1]. These candidates must simultaneously (i) conform to syntactic constraints and (ii) satisfy requirements defined as semantic properties (uniformly called multi-paradigm consistency criteria). Violating these constraints and consistency criteria can result in *inconsistencies*. Such inconsistencies can be resolved by changing the underlying model through fix operations [2].

Validating static well-formedness constraints, assuring semantic properties (e.g., correctness, completeness, determinacy) and evaluating extra-functional properties (e.g., availability, throughput) requires complex analysis techniques which exploit fundamentally different abstractions and solver technologies. Existing DSE techniques are limited to sequentially calling independently operating solvers. For instance, the generic DSE framework presented in [3] supports arbitrary analysis tools, but requires a predefined workflow. Similarly, Phoenix Integration ModelCenter [4] is a framework enabling DSE of a variety of parameterized models by calling external solvers in a pre-determined sequence. Both approaches are limited in that explicit dependencies between solvers cannot be specified. Therefore, violations to constraints imposed by a previously called

solver cannot be detected by subsequently called solvers, thereby potentially carrying forward and further exploring infeasible solutions.

This paper proposes an alternative approach based on performing semantic fixes to resolve diverse, multi-paradigm inconsistencies through semi-automated orchestration of existing checker and solver tools that are driven by a guided DSE process. The paper is a result of a collaborative effort of the authors at the 2014 Computer Automated Multi-Paradigm Modelling (CAMPaM) workshop. Our hypothesis is that focusing on orchestration and decomposition of the overall *design space exploration problem* (DSEP) is a necessary step towards managing the complexity associated with running expensive analyses in scenarios where multiple DSEPs must be solved simultaneously. Our proposed solution is an orchestration strategy that uses heuristics to define the data and control flow between the different analysis and exploration phases to call the appropriate off-the-shelf solvers as needed. Core is the idea of back-tracking solutions automatically by exploiting the dependencies of underlying solvers, the multi-paradigm consistency criteria, and the orchestration strategy.

The paper is organized as follows: we introduce our view on DSE and develop our conceptual idea through application to an example problem in Section 2. Our proposed approach is discussed in further detail in Section 3.

2 Orchestrating Design Space Exploration Problems

DSE is the process of generating and analyzing a set of design alternatives for the purpose of finding the most preferred configuration. In our framework we consider a *guided* incremental approach to exploration, in which potential *design candidates* (alternatives) must meet a set of *constraints*. Constraints can apply to both *numerical* and *structural* attributes [1]. Design candidates are evaluated based on quality metrics such as cost or dependability (*optimality criteria*). *Hints* are provided in the form of heuristics to guide the design process, which mitigates the prominent issue of most DSE approaches: their inefficiency in handling structural constraints and dynamic manipulation of elements due to the use of model checking in conjunction with exhaustive state space exploration [1]. Hints represent expert knowledge (i.e., *heuristics*) that can guide the exploration process by detecting dead ends or unpromising paths early.

2.1 Semantic Inconsistency Fixing as a DSEP

Within the context of our work, we view the constraints typically used for identifying feasible solutions in a DSE as *semantic consistency constraints*. A violation of these is indicative of an inconsistency. We say that semantic consistency constraints belong to either the class of *solution* or *path constraints*. One prime example of solution constraints are constraints used to ensure adherence to language syntax and structural semantics. Path constraints represent a set of conditions that valid exploration paths must fulfill; they are used to determine whether, as a result of applying an operation, the intended model functionality is preserved. Through path constraints, pruning of exploration paths enables to focus on optimizing only valid paths.

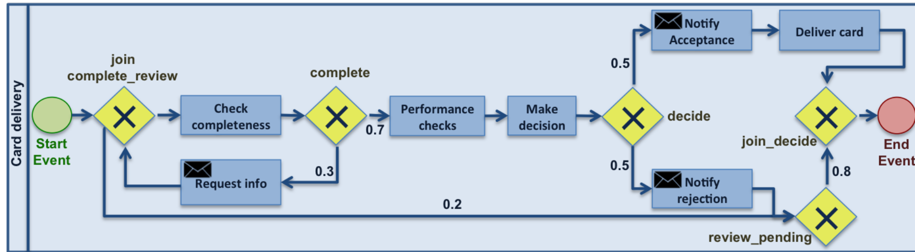

Fig. 1. A BPMN model for a card delivery system

Semantic inconsistencies can be identified by specifying paradigm-specific solution and path constraints. Given a graph-based model representation, *negative graph constraints* can be used for the purpose of checking static semantics [2]. In our framework, we associate these negative constraints with one or more *fixing operations* that can be used to alleviate the respective semantic inconsistency. Each fix is performed in two steps: first, an analysis is performed by a *checker*, which looks for the existence of possible inconsistencies with respect to desired characteristics. In a second step, the identified inconsistencies are *resolved*. The resolution is performed based on a set of given model transformation operations and is guided by i) a given optimality criteria, and ii) heuristics and hints. Dependencies between fixing operations are provided in the form of (formally captured) hints. These dependencies can then be used for early pruning of non-optimal solutions. For instance, if the optimization criteria implies a solution with the smallest difference to the original model, a dependency analysis of the model operations allows an a-priori determination of which combination of operations should be applied in what particular sequence [2].

2.2 Illustrative Example

In the following, we illustrate how different DSEPs can be orchestrated to find and resolve multi-paradigm semantic inconsistencies. For this purpose, a process model – more specifically, a *business process model* (conforming to the *Business Process Model and Notation* (BPMN) [5]) – is used. A business process model describes the set of activities that an organization should implement and execute to provide a particular service or product.

Figure 1 illustrates an example BMPN model for a card delivery process of a service company (only one lane is shown due to space constraints). The company checks if the client filled out the forms correctly and continues to process the order until the card is delivered. Because of production constraints or malformed requests, the order can be rejected.

When modeling this process, it is possible for a number of syntactic and semantic inconsistencies to be introduced. For example, one well-formedness constraint of BPMN2 states that a diverging OR-gate has to be preceded by a single decision activity to specify what gate has to be taken [5]. Additionally, a deadlock can be introduced if two processes in different lanes are each waiting for messages that are only sent after the respective waiting tasks have terminated. Both the detection and the fixing of these types of inconsistencies is non-trivial.

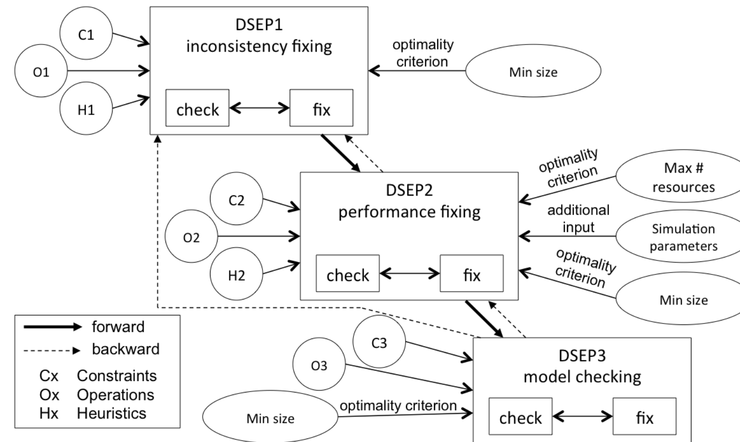

Fig. 2. Example orchestration scenario

In addition to these inconsistencies, it is often desired to optimize business processes. This requires exploration of the design-space of possible alternative processes that result in the same service or product (i.e., that are functionally equivalent). One possible optimization is the redistribution of resources to reduce the expected waiting times at some of the activities.

Exploring the design space requires all of these inconsistencies to be identified and fixed – ideally, in an automated fashion. Figure 2 illustrates one possible orchestration of solvers that check and fix the aforementioned inconsistencies. The first of the three DSEPs manages conformance to language well-formedness constraints, the second DSEP optimizes the performance of the model. Finally, the third DSEP performs a formal verification through model checking to check that deadlocks cannot occur. Inconsistencies are checked for by evaluating constraints (Cx) and fixed using a particular transformation operation (Ox).

In our example, the *performance fixing* DSEP is performed by first transforming the BPMN model to an extended queuing network [6]. The following types of performance fix operations are possible: (1) Resource Distribution: Resources are redistributed to each of the activities resulting in shorter or longer service times. (2) Parallelization: Activities that are executed in sequence can be parallelized. (3) Serialization and Deserialization: An activity is divided into multiple activities or merged into a single activity.

When executing different DSEPs sequentially, inconsistencies may be introduced. For instance, when a *parallelization* rule of the performance fixer is executed, a violation of a well-formedness constraint may occur: e.g., the activities *check performance* and *make decision* should not be parallelized because a diverging OR-node is connected to the decision activity. Therefore, if the rule is applied for either activities, the model violates the constraints checked by the first solver. Capturing such dependencies among fix operations formally as heuristics (Hx) across different solvers enables automated backtracking to previous DSEPs whenever necessary.

3 Discussion

Our illustrative example revealed several general aspects of the DSEP orchestration problem for inconsistency fixing. First, it highlighted the fact that DSE orchestration can be viewed as a decomposition problem: how to split a DSEP into a set of sequential and parallel smaller DSEP steps that, together, are less expensive than the original DSEP? Second, the ability to backtrack within an orchestration is essential. Third, an orchestration can be modeled using a modeling language with specialized semantics.

3.1 Decomposition of a DSEP

There are several dimensions along which a DSEP decomposition can occur.

Submodel decomposition. The model being fixed can be decomposed into submodels and thus the DSEP is split into an orchestration of a set of DSEP's over the submodels. For instance, in our example we could have decomposed the card delivery system model into separate submodels for each lane. Non-overlapping submodel DSEP's could then be executed in parallel in the orchestration. For overlapping models, interface automata [7] could be defined.

Abstraction decomposition. If there is a natural way to create abstractions of the model being fixed, the DSEP can be orchestrated into a sequence of increasingly refined DSEP's. For example, BPMN models can be abstracted by collapsing a portion of the model into a single Activity thus producing a more abstract model. A solution to the DSEP for the abstract model can be used to constrain the DSEP for the original model by limiting its search to a subspace. If no solution is found in the subspace, backtracking occurs to the abstract DSEP to find another abstract solution.

Constraint decomposition. Decomposing the set of solution constraints yields an orchestration into a sequence of DSEP's, each using a subset of the constraints. This is the kind of decomposition we used in the card delivery system example. If a solution is found in one step, it is passed as the initial model to the next DSEP step and so on. This kind decomposition is most effective if the fix operations of each step cannot cause a violation of the constraints in any of the previous steps. If this condition cannot be guaranteed, then the constraints of the first DSEP must be rechecked on a solution to the second DSEP and if a violation occurs, backtracking must be performed. Constraint decomposition is particularly useful when different subsets of constraints require different solvers (as is the case in our example). In this case, the constraints requiring more expensive solvers can be put later in the sequence.

3.2 Backtracking

As discussed above, some decomposition approaches require support for backtracking. This is required to ensure that the orchestration is *complete* - i.e., that every solution of the original DSEP is reachable by the orchestration.

In contrast to the related work discussed in the introduction that define specialized and explicit backtracking mechanisms for their DSE problems, we propose a *general automated and implicit* backtracking strategy. This is possible

because we are restricting our attention to the problem of fixing inconsistencies and all DSEPs in an orchestration have a set of fix operations. An automated analysis of the dependencies among fix operations in the different DSEP's can determine the points at which backtracking is necessary. This is similar to what is known as *dependency-directed back-tracking* [8] in artificial intelligence. In our case, this entails the ability to automatically backtrack by exploiting known dependencies among underlying solvers and multi-paradigm consistency criteria.

3.3 Towards an orchestration modeling language

Our goal with this work is to implement automated backtracking semantics on top of an orchestration modeling language used for defining the forward flow of the orchestration. For example, a designer may use a language such as UML activity diagrams to define the forward flow orchestration of a set of DSEP's. Automated dependency analysis would be used to discover dependencies between these DSEP's and then when the orchestration is executed, this would the trigger automatic backtracking where necessary.

4 Conclusions & Future Work

This paper introduces a conceptual basis for orchestrating multi-paradigm design space exploration problems (DSEPs). A core idea is the decomposition of the overall DSEP by exploiting dependencies between model operations. By allowing for backtracking, the overall cost of multi-paradigm DSEPs can be decreased significantly. However, in our approach, this is a conclusion that can only be reached under the assumption that dependencies among model operations can be established – i.e, under the assumption that a common formalism for representing and manipulating models can be identified.

We believe that the semi-automation of the described solution is technically viable, and, for achieving it, further research will be carried out. Future work should include the development of a language for modeling DSEP orchestrations. Additionally, the concepts discussed in this paper could be extended to a *dynamic* orchestration where, similar to co-simulation, a control component manages interactions between DSEPs rather than relying on manually defined backtracking paths.

References

1. Abel Hegedus, Akos Horváth, István Ráth, and Dániel Varró. A Model-Driven Framework for Guided Design Space Exploration. In *Procs of ASE*, pages 173–182. IEEE Computer Society, 2011.
2. A. Hegedus, A. Horváth, I. Ráth, M. C. Branco, and D. Varró. Quick Fix Generation for DSMLs. In *Procs of VL/HCC*, pages 17–24. IEEE, 2011.
3. T. Saxena and G. Karsai. MDE-Based Approach for Generalizing Design Space Exploration. In *Procs of MODELS*, pages 46–60. Springer, 2010.
4. Phoenix Integration ModelCenter. `http://phoenix-int.com`.
5. Business Process Model And Notation (BPMN) Version 2.0, January 2011.
6. P. Bocciarelli and A. D'Ambrogio. Automated Performance Analysis of Business Processes. In *Procs of TMS/DEVS*, 2012.
7. L. De Alfaro and T. A. Henzinger. Interface Automata. *ACM SIGSOFT Software Engineering Notes*, 26(5):109–120, 2001.
8. R.M. Stallman and G.J. Sussman. Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis. *Artificial Intelligence*, 9(2):135–196, 1977.